Lab 9: Designing the PeaPOD User Management System with MVC

❖ General Guideline

This team project is designed to demonstrate the ability of collaboratively developing complex software systems with the most popular software design pattern: MVC. Each team may freely choose up to 4 students from the class. It is important that everyone actively participates and collaborates with others in the team. Each member of a team must be responsible for completing at least two classes. Put your name in PHP comments in the files you complete. Each team needs to turn in only one copy.

❖ Project Description

The **PeaPOD User Management System** allows management of user accounts. It supports the basic user management tasks you may find in most user-centered applications. Main features include user registration, login, logout, and password reset. Data of the application is stored in a MySQL database named **usersystem**. The database has one table named **users**.

You must design the **PeaPOD User Management System** with the MVC pattern. Routing should be done using query string variables, similar to the ToyMVC application. Don't use RESTful urls for routing in this lab.

Be sure you test the complete application at https://i211.sitehost.iu.edu/index.php?u=4

❖ Application Logic and Organization

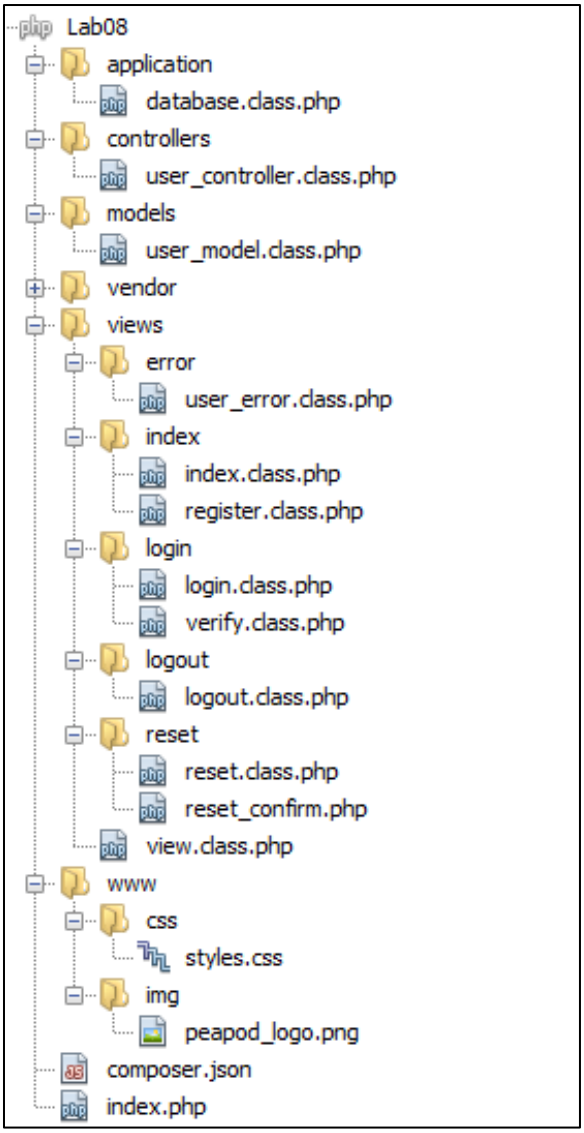
Organize your files according to the MVC principles:

1. **Models**: all files that deal with data and business logic.
2. **Views**: presentation files. Create a separate folder for all views related one particular action.
3. **Controllers**: files that contain a set of methods to handle different actions.

A bootstrap file (*index.php*) should be placed in front of the controllers. All requests are sent to the bootstrap file. Every request contains a query string variable named **action**. There are eight possible values for this variable: **index**, **register**, **login**, **verify**, **logout**, **reset**, **do_reset**, and **error**. These values corrspond to the eight actions the application handles:

- **index** – display the registration form; default action.
- **register** – register the user by creating a user account and storing the data in the database.
- **login** – display the login form.
- **verify** – verify username and password against a database record.
- **logout** – log a user out of the system.
- **reset** – display the password reset form.
- **do_reset** – reset the password by updating a database record.
- **error** – display the custom error page.

The following diagram shows the file and folder hierarchy of the entire application once it is complete:



Complete files that have been provided in the download:

1. *www/css/style*.css: the external style sheet for the entire application. You may add additional styles if you wish.
2. *www/img/peapod_logo.png*: the logo image file.
3. *composer.json:* json file for composer.
4. *usersystem.sql*: the sql file for the **usersystem** database. It will also create the table named **users** and populate it with one user account: *admin*, *password*. This file can be removed once the database has been imported into your MySQL server.

5. *application/database*.class.php: the **Database** class. Be sure you modify the login and password to use an existing account on your machine.
6. *views/view.class.php:* The **View** class defined in this file is the parent class of all view classes. It defines two methods: **header** and **footer**. Calling these methods in individual view classes will insert code into the top and bottom of the view pages. For individual view classes, you will only need to add page specific content.
7. *views/error/user_error.class.php:* one of the view files. The **UserError** class extends the **View** class. The **display** method defined in the class displays an error message. The page specific content consists of three div blocks: top row, middle row, and bottom row. Please refer to the structure of this file when creating other view files.

Files that need to be completed or created:

8. *index.php:* the homepage of the application. It also contains script to dispatch requests. You need to add code to retrieve the **action** (a query string variable) value and then invoke a proper method defined in the **UserController** class with a **UserController** object. This file has been provided and needs to be completed. For security, data retrieved from a form must be properly sanitized.
9. *controllers/user_controller.class.php*: this file defines a class named **UserController**. Need to define a method for each of the eight actions described in the previous **Application Logic and Organization** section. Name each of these methods the same as the action that the method handles.
10. *models/user_model.class.php*: this is the application model. This file defines a class named **UserModel** with four public methods.
    - **add_user:** retrieve user details from the registration form and then add them into the **users** table in the **usersystem** database. The method returns true if the insertion is successful or false if it fails. Passwords need to be hashed before they are stored into the database. To hash a password, call **password_hash** function. Please refer to http://php.net/manual/en/function.password-hash.php.
    - **verify_user:** retrieve a user's username and password from the login form and then verify them again a database record. If both username and password are valid, create a temporary cookie to store the username and return true; If either username or password is invalid, return false. To verify a hashed password, call **password_verify** function. Please refer to http://php.net/manual/en/function.password-verify.php.
    - **logout**: logout a user by destroying the temporary cookie created when the user signs in. The method should return true.
    - **reset_password**: retrieve a user's username and password from the password reset form and then update the user's password in the database. Make sure the password is hashed before it gets updated. Return true if the password is successfully updated or false otherwise.

View classes for registration:

11. *views/index/index.class.php*: the view file for the application homepage. It defines a class named **Index** and a public method named **display**(), which displays the registration form. The registration form allows a user to enter username, password, email, first name and last name. Please note the following regarding the form:
    - Use HTML5 validation techniques to ensure all fields are required; the email field is filled with an email address; and the password field is filled with 5 or more characters.
    - Use "post", but not "get", method to submit form data.
    - The form should be sent to *index.php* along with the querystring variable called **action**.
12. *views/index/register.class.php*: display a confirmation message after a user has attempted to create an account. Please note the page should display different messages indicating whether the registration is successful or failed and then display hyperlinks accordingly.

View classes for login:

13. *views/login/login.class.php*: the view file for displaying the login form.
14. *views/login/verify_user.class.php*: the view file for displaying a confirmation message after a user has attempted to log in. The page should display different messages showing whether the login is successful or failed and then display hyperlinks accordingly

View class for logout:

15. *views/logout/logout*.class.php: display a conformation message after the user has logged out.

View classes for password reset:

16. *views/reset/reset.class.php*: display the password reset form if the user has logged in. The form should automatically fill the username field using the cookie created when the user logs in. It should also make the username field read-only. The form should use POST method to submit data. If the user is currently not logged in, display an error message instead.
17. views/reset/reset_confirm.class.php: display a conformation message indicating whether the user's attempt to reset password was successful or failed. It should also display hyperlinks accordingly.

❖ Instructions

Here is my suggestion on how you should divide the project and split the responsibility among team members:

Member 1: index.php, user_controller.class.php
Member 2: user_model.class.php
Member 3: all view classes

❖ Code style guidelines

1. Code style is as important as the code itself. Code style includes enough comments, adequate space between code blocks, and proper indentation, etc. Read details and view sample code from http://pear.php.net/manual/en/standards.php.
2. Please provide sufficient comments in your source code. Source code is a language for people, not just for computers. Comment as you go and don't wait for later. Ask yourself: "How will the next person know that?" Commenting code shows your professionalism, but also helps your grader understand your code.
3. Every class file should contain a header in this format:

```
/*
* Author: your name
* Date: today's date
* Name: file name
* Description: short paragraph that explains what the class is about
*/
```

4. Indent your code. Leave enough space between code blocks. You can always use Alt+Shit+F in NetBeans to automatically format your code.

❖ Turning in your lab

Each group needs to turn in one copy. Provide names of all members in the team in Canvas. Your work will be evaluated on completeness and correctness. Thoroughly test your code before you turn it in. It is your responsibility to ensure you turn in the correct files. You will NOT receive any credit if you turn in the wrong files whether or not you've completed the lab.

1. Zip the entire **Lab09** folder and save it as *Lab09.zip*.
2. Upload the *Lab09.zip* file in Canvas before the lab's deadline.

❖ Grading rubric

Your TAs will assess your lab according to the following grading rubric. You should very closely follow the instructions in this handout when working on the lab. Small deviations may be fine, but you should avoid large deviations. You will not receive credits if your deviation does not satisfy an item of the grading rubric. Whether a deviation is small or large and whether it satisfies the requirement are at your TAs' discretion. For the convenience of grading, the total point of this lab is 60. To calculate each member's score, divide the project score by 3. Here is the breakdown of the scoring:

Creating files (51 points)

Activities	Points
Complete index.php	5
Create user_model.class.php	10
Create user_controller.class.php	15
Create index.class.php	3
Create register.class.php	3
Create login.class.php	3
Create verify.class.php	3
Create logout.class.php	3
Create reset.class.php	3
Create reset_confirm.class.php	3

Programming style (9 points)

Activities	Points
Comment your code	6
Use white spaces to separate code sections	1
Indent and line up code	2